Section 4
System Hardware and Software

This section describes the hardware and software of the HP/Apollo DOMAIN computer system on which the Enhanced Traffic Management System (ETMS) was developed and on which it operates. The description is neither complete nor in depth, but provides a brief overview of the ETMS computing environment, highlighting those system features that significantly affect the ETMS software design. (HP/Apollo provides an extensive set of documentation covering all aspects of the system.)

The DOMAIN system provides an environment of distributed workstations (commonly referred to as *nodes*) connected through a high-speed Local Area Network (LAN) and accessible through a network-wide, virtual operating system. Each node can operate as a stand-alone computer. The connection of the nodes (physically through the LAN and virtually through the operating system) also permits any user or application to use all or part of the total network resources (processors, memory, disk space, peripherals). The distributed approach, when implemented correctly, provides great flexibility in the design and development of a software system; as the system requirements change, the available hardware resources can be expanded or changed incrementally. In addition, the operating system makes data communications between the functions distributed within the network easy to implement, efficient, and reliable.

4.1 System Hardware

This section describes important aspects of the HP/Apollo hardware that runs the ETMS: the types of HP/Apollo nodes used, their configuration, and the characteristics of the LAN.

4.1.1 Node Types

The ETMS relies primarily on two types of nodes for its operation: the DN4000/4500 and the HP433. The DN4000/4500s are used to handle the communications between the Volpe Center and the FAA sites. The HP433s are used for computation-heavy processes (traffic modeling) and for graphics-related processing (the *Aircraft Situation Display* [ASD]).

The DN4000 uses the 25MHz, 32-bit MC68020 central processor and the MC68881 floating point co-processor. The DN4500 was upgraded to the MC68030. Each DN4000/4500 used in the ETMS is configured with the maximum available memory size (32 megabytes) and the largest available hard disk (330 Megabytes for the DN4000 and 660 Megabytes for the DN4500). The DN4000/4500 allows up to 1024 concurrent processes, each of which can

address up to 1 gigabyte of virtual memory. Each DN4000/4500 is equipped with a standard keyboard (typically with a mouse) and a 19-inch, high-resolution, black-and-white display.

The HP433 uses a 33MHz, 32-bit MC68040 central processor and 128K cache memory. Each HP433 is configured with 64 Megabytes of physical memory (expandable to 128 Megabytes) and a 2 Gigabyte hard disk. Each HP433 is equipped with a 19-inch, high-resolution, 1280 x 1024, color graphics display with eight built-in planes of display memory and 256 simultaneously available colors. The HP433 allows up to 1024 concurrent processes, each of which can address up to 1 Gigabyte of virtual memory. Each HP433 is equipped with a standard keyboard (typically with a 3-button trackball). Each HP433 is also equipped with two interfaces: an Apollo Token Ring and an Ethernet 802.3.

4.1.2 Local Area Network

The HP/Apollo DOMAIN nodes are connected at a local site through a proprietary HP/Apollo LAN, using a ring structure and a token-passing protocol. The network provides an inter-node communication speed of 12 million bits per second and supports up to 1,000 nodes on a single network loop, with a maximum inter-node distance of 1,000 feet.

4.2 System Software

This section describes the key features of the DOMAIN system software used to develop and execute the ETMS:

- Domain/OS operating system
- Programming languages
- Domain Software Engineering Environment (DSEE)
- System routines for
    - Graphics
    - Mailboxes and Sockets
    - Mapped files
    - Mutual exclusion locks
    - Eventcounts
    - Process management
    - Miscellaneous functions

4.2.1 Domain/OS Operating System

The Domain/OS is an operating system developed by HP/Apollo. It allows a user or application to take full advantage of the hardware network by providing access to the network resources at the operating system level. Domain/OS allows a user to execute multiple processes on any node of the network, with a total limit of 1024 processes on any individual node. Each process runs in virtual memory the size of which (from three megabytes to two gigabytes) depends on the type of node on which it is executing. The virtual processes utilize physical memory through a proprietary demand paging algorithm.

Domain/OS also allows any user or process to access any disk storage device on the network in a virtual manner. Each disk on the network is identified by a logical name. UNIX-like, tree-structured directories are used to organize the files on the disks. The directories can span any number of physical locations on the network since a *leaf* of a tree on one node can be linked to a directory on another node.

Domain/OS can be configured in one of three environments: AEGIS, UNIX SysV, or UNIX BSD. ETMS nodes are configured to use the AEGIS environment, which allows them to take the fullest advantage of the HP/Apollo network.

Domain/OS supports multi-windowing through the DOMAIN Display Manager. The Display Manager allows users to create windows for executing processes (UNIX-like shells) or for editing files (edit pads). Processes also create windows for input or output. Both users and/or processes can flexibly control the positioning of windows on the display.

Domain/OS also supports multi-windowing through X Windows and the Motif Window Manager. As with the DOMAIN Display Manager, windows can be created and manipulated by users or processes. UNIX editors, such as vi, can be used to edit files within the X Windows environment. Domain/OS can be configured to run either the DOMAIN Display Manager or the X Windows system, or both simultaneously.

4.2.2 Programming Languages

HP/Apollo supports a number of languages in the DOMAIN environment. ETMS was developed almost exclusively in the Pascal programming language, chosen for its ease of maintainability. The HP/Apollo version of Pascal features many extensions to standard, classroom Pascal-extensions which resolve many of the standard version's practical limitations. The most significant is the ability to develop source code in *modules* that may be compiled separately and later bound into an executable program.

HP/Apollo also supports the C programming language. A small amount of C code was written to perform the semantic parsing of the flight path fields from the NAS messages. C was chosen for the job because its superior string handling capabilities made design and implementation of code more straightforward and efficient. The C functions are accessed through Pascal subroutine calls which transfer control to the appropriate C routines. Since the initial implementation, several major functions (SDB, Parser, EDCT) have been converted from Pascal to C.

4.2.3 DOMAIN Software Engineering Environment

The DSEE is a software and database management system for supporting a software engineering project development effort, particularly in a team environment. DSEE supports developers in the following ways:

- Maintaining and tracking successive versions of source code modules.
- Representing the dependency relationships between source code modules and object code.
- Notifying developers automatically of changes to source code modules.
- Coordinating simultaneous development of common modules between developers.

DSEE supports configuration management in the following ways:

- Tracking source code histories.
- Allowing complete recoverability of old versions of systems.
- Performing organized, automatic releases.

DSEE is fully integrated with the DOMAIN system to allow editors, compilers, and other programs to gain access to the source code managed by DSEE and to allow the development environment to spread throughout the network.

DSEE consists of five types of *software managers* performing different functions. Three of these managers are significant in the ETMS development: the history manager, the configuration manager, and the release manager.

(1) *History Manager* — handles the source code. Code is developed in DSEE *elements* (corresponding to files) organized in DSEE *libraries* (corresponding to directories). The history manager maintains every version of an element throughout its development history. (The versions are stored internally by DSEE as incremental changes, providing very compact storage.) An element must be reserved to be worked on. (This prevents other developers from changing it simultaneously.) The developer is also required to enter comments explaining why the element was reserved.

The history manager allows *branches* in the *line of descent* of an element, which supports simultaneous development in a controlled manner. Separate branches can be continued indefinitely or can be merged together using DSEE merge facilities. Branches must be named. Versions within a line of descent are numbered by DSEE, but the developer can also name them for easy identification.

(2) *Configuration Manager* — provides automatic and controlled building of object code (executable programs) by use of the *system model* and the *configuration thread*.

(a) The system model is a blueprint for building a program. It defines the elements that must be compiled and linked to create an object, and where they can be found. The system model also defines the dependencies that exist between elements and objects. (If an element is changed, which other elements should be re-compiled and which objects rebuilt?)

(b) The configuration thread separately defines the versions of each element for the build.

The system model and configuration thread thus allow separate control over program structure and version. When DSEE builds a program, it records the versions of each element used in the build and saves all the intermediate object code as well as the final result. When rebuilding a program, DSEE reuses the object code, as long as no dependent elements or objects have been changed. This provides more efficient builds, particularly when several developers are working on one program.

(3) *Release Manager* — is used when a release of a system is made. The release manager uses the information recorded during the build of the system to determine exactly which version of each system component should be released. The release manager sends all source code; build tools (compilers, linkers, etc.), and the executable code from DSEE to a location identified by the developer. The release manager thus allows a complete, rebuildable version of a software system to be generated and saved outside of DSEE for safekeeping or for delivery to a customer.

Five DSEE commands are used to create an executable release version of each DSEE-maintained program: set system**,** set model**,** set thread**,** build**,** and create release.

(1) *Set system* — In the DSEE environment, a system is a directory that contains references to the software modules used in an executable program. Each program built in the DSEE environment must have a system directory acting as a database for the build. All information and objects needed during a build are stored in this directory. The set system command defines the current system and conveniently restores the work settings (the current system model and library) to those last used.

(2) *Set model* — A system model tells DSEE how to construct a particular program. It specifies the source code, tools, and building procedures required to compile and bind a program. The set model command selects the model to be compiled and worked on during the development session.

(3) *Set thread* — The set thread command and its options define and validate the current configuration thread. The configuration thread (a list of rules written in the configuration thread language) specifies which versions of the files listed in the model are to be used in the build.

(4) *Build* — Once the current system, current system model, and current configuration thread have been set, the developer is ready to build the program. When DSEE builds a component, it draws on the system directory, system model, and configuration thread to construct the *specification.* The specification determines which element versions, regular files, tools, translation rules, and options are needed to build the component. DSEE searches through the stored pool of previously built components-reusing existing components before building new ones. The results of the build are then stored in this binary pool so that with each modification, only the altered components have to be rebuilt.

DSEE also generates a *build map* that documents the specification in a readable form. It contains information such as the identification of the developer, the system model setting, the translation rules, the elements versions, and the files used in the build.

(5) *Create release* — When a system or system component release is declared, DSEE copies all object code to a release directory accessible for general use. The *create release* command safeguards all or part of a previously built program. The command declares the specified build, assigns a name to the release, creates a release directory, copies the release's derived objects and specifications to the release directory, and creates another user-readable build map documenting each derived object in the release.

4.2.4 System Routines

The DOMAIN system routines are special-purpose subroutines (provided by HP/Apollo) that may be used with any of the DOMAIN supported programming languages. These routines are invoked by pre-defined subroutine calls. The object code for the system routines is automatically included at the time that a program is built.

4.2.4.1 Graphics

The DOMAIN system provides several levels of routines for interacting with graphics hardware. The ETMS utilizes the group of graphics routines known as the *Graphics Primitives Resource (GPR),* a set of fairly low-level graphics functions. The use of *GPR* places much of the burden of the graphics programming on the application, but allows the graphics hardware to be customized for better performance. The *GPR* routines are fully integrated with the DOMAIN Display Manager; this allows customized graphics to be generated within the normal, flexible, multi-windowing environment.

The ETMS uses *GPR* to generate the graphics for the *ASD*. The ETMS functions that have been converted to C use the X windows system.

4.2.4.2 Mailboxes and Sockets

The mailbox routine is a facility for inter-process communications.

(1) A process can create a mailbox.

(2) The process can then become the *server* to the created mailbox.

(3) Other processes can connect as *clients* to the mailbox.

(4) The servers and clients can then transmit information between them through mailbox *channels*.

The mailbox allows servers and clients to be on the same or separate nodes. Mailboxes are identified by logical names, allowing the physical locations of the servers and clients to be transparent to each other.

The ETMS uses mailboxes for communications among the various ETMS functions at each site. A mailbox is implemented to ensure that no messages are lost.

ETMS functions that have been converted from Pascal to C use TCP/IP sockets rather than mailboxes for interprocess communication. Because ETMS operates in a mixed environment, bridge programs are used to convert data from one protocol to the other.

4.2.4.3 Mapped Files

Mapping routines associate a disk file with an area of a process virtual memory. Once a file is mapped, a process can gain access to any part of it by providing the address of the desired data. With files of fixed length records, the address of any record can be computed. The operating system gets the necessary pages of a file into physical memory through the normal demand paging mechanism. Multiple processes can share mapped files, although for write access, the process must be on the same node. The mapping routines do not detect or resolve contention between multiple processes.

Mapped files are a particularly useful feature in Pascal programming because they allow quick, random access to large files that would normally be read sequentially. Mapped files have other advantages:

(1) When the process that mapped the file terminates (either normally or through a software interrupt), the final contents of the file are paged back to the disk.

(2) The process can then remap the file when it restarts and not lose any data.

The ETMS uses mapped files to implement all the traffic modeling databases and the buffers for the buffered inter-process communications routines.

4.2.4.4 Mutual Exclusion Locks

A mutual exclusion (mutex) lock routine coordinates processes that access a shared resource:

(1) A mutex lock can be defined by the sharing processes and associated with a resource, such as a mapped file.

(2) The processes can then be implemented so possession of the mutex lock is required to access the resource.

(3) The routines ensure that only one process has possession of the mutex lock at a time.

The ETMS uses mutex locks to control shared data access in the *Generic Buffering Package* (*GBP*).

4.2.4.5 Eventcounts

Eventcount routines can be used to synchronize processes with each other or with system events. An eventcount is a value that can be incremented or decremented at the occurrence of some event. The event can be system-defined (such as an I/O operation or clock time) or an event within an application program. Processes can wait for events until the eventcount reaches some value.

The ETMS uses eventcounts in many places, including the periodic generation of data for the *ASD* and synchronization of the traffic modeling processes that use buffered inter-process communications.

4.2.4.6 Process Management

The process management routines allow a process to invoke and monitor other processes. The process which does the invoking is known as the *parent*, while the processes invoked are known as the *children*. A parent process can start the execution of a child on the same or a different node. The parent can wait for the child to be successfully started and can then re-check the status of the child during execution.

Process management routines are useful when the execution of one process is directly dependent on the execution of others. When a process directly invokes other processes, it knows explicitly that they are executing before it begins executing itself.

The ETMS uses the process management routines for the traffic modeling functions. The I/O processing for these functions is performed by separate processes that run concurrent with the data processing and ensure that each node's processing resources are fully utilized. The I/O processes (without which the data processing is impossible) are started up as children to the main process.

4.2.4.7 Miscellaneous

Other types of useful system routines have been implemented throughout the ETMS but do not significantly affect the design approach. These are as follows:

(1) Display Manager Routines - allow a process to create and manipulate windows for showing text output on the display screen.

(2) Time Routines- allow a program to access the current time and date from the operating system and to perform conversions between different time formats.

(3) Variable Formatting Routines- allow data to be converted from one Pascal data type to another. Variable formatting is used to process inputs that may be in an undetermined format and to transmit variable length records between ETMS processes.